Formally Verified Computation of Enclosures of Solutions of Ordinary Differential Equations

Fabian Immler*

Institut für Informatik, Technische Universität München
immler@in.tum.de

Abstract. Ordinary differential equations (ODEs) are ubiquitous when modeling continuous dynamics. Classical numerical methods compute approximations of the solution, however without any guarantees on the quality of the approximation. Nevertheless, methods have been developed that are supposed to compute enclosures of the solution.
In this paper, we demonstrate that enclosures of the solution can be verified with a high level of rigor: We implement a functional algorithm that computes enclosures of solutions of ODEs in the interactive theorem prover Isabelle/HOL, where we formally verify (and have mechanically checked) the safety of the enclosures against the existing theory of ODEs in Isabelle/HOL.
Our algorithm works with dyadic rational numbers with statically fixed precision and is based on the well-known Euler method. We abstract discretization and round-off errors in the domain of affine forms. Code can be extracted from the verified algorithm and experiments indicate that the extracted code exhibits reasonable efficiency.

Keywords: Numerical Analysis, Ordinary Differential Equation, Theorem Proving, Interactive Theorem Proving

1 Introduction

Ordinary differential equations (ODEs) are used to model a vast variety of dynamical systems. In many cases there is no closed form for the solution, but one can resort to numerical approximations. They are usually given by "traditional" one-step methods like the Euler method or the more general family of Runge-Kutta methods, which approximate the solution in several discrete steps in time. However, especially in safety-critical applications, approximations are too vague in that they provide no rigorous connection to the solution.

To establish such a connection, in the area of "guaranteed integration", different approaches have been proposed. They have in common that they do not compute with approximate values, but with sets enclosing the solution. The most basic way to compute with sets is interval arithmetic, which suffers from the wrapping effect (i.e., large overapproximations when enclosing rotated boxes

* Supported by the DFG Graduiertenkolleg 1480 (PUMA)

in a box) and cannot track dependencies between variables. The proposed approaches differ in the data structures that represent the sets as well as the algorithms that are used to compute them.

A well-studied family of algorithms is based on on Taylor series expansions and computing with interval arithmetic, surveyed e.g., by Nedialkov [18], and implemented in tools like AWA [15], ADIODES [25], VNODE [17], VNODE-LP[19]. Those overcome the wrapping effect with QR-decomposition. A different approach is based on Taylor models, which suffer from neither the wrapping effect nor the dependency problem and which were studied by Makino and Berz and implemented to solve ODEs in COSY [1]. A survey of Taylor model based methods is given by Neher *et al.* [20]. Tucker [26] uses the "traditional" Euler method with interval arithmetic and uses interval splitting to overcome the wrapping effect. Bouissou *et al.* [3] also use "traditional" methods, but they represent sets with affine forms [6] to overcome the wrapping effect and track linear dependencies.

Most of the aforementioned "guaranteed" methods have in common that the proofs that they actually compute enclosures of the solution are carried out on a relatively high level, without formal connection to the source code. Nedialkov [19] has been worried about this gap and responds with implementing VNODE-LP using literate programming, such that the correctness can be verified via code review by a human expert. The operations used in COSY are (manually) proved correct in [24] – but only for the basic operations on Taylor models, without a connection to ODEs.

Our work aims at narrowing the gap between implementation and proof even more by allowing for mechanical software verification. We formalize both proof and algorithm in an interactive theorem prover, namely Isabelle [22]. The theorem prover provides a formal language to express mathematical formulas and allows to prove theorems in a rigorous calculus, where every reasoning step is checked by the system. Isabelle/HOL implements higher-order logic, a subset of which can be seen as a functional programming language. Isabelle/HOL therefore allows to extract code from the formal specifications.

Our approach is to give a specification of a guaranteed method for ODEs in Isabelle/HOL. We verify that the computed enclosures are correct with respect to a formalization of ODEs in Isabelle/HOL. Our specified method is executable, we therefore extract code and compute enclosures for some examples.

The method we chose to formalize is based on the approach taken by Bouissou *et al.* [3]: they essentially let "traditional" methods operate on sets represented by affine forms. We liked the flexibility of their framework – the fact that it can be extended with different "traditional" methods, which are all well-studied and each known to be suited for particular kinds of ODEs. Moreover, we had already formalized a (rudimentary) numerical analysis of the Euler method.

1.1 Contributions

We contribute a formal and mechanically checked verification of a set-based Euler method. We therefore provide a formalization of affine forms, a formal

specification of the Euler method based on affine forms and a formal correctness proof with respect to the formalized mathematical specification of ODEs. In the course, we discovered subtle issues in informal proofs given for other set-based methods (see also Section 2.4).

Note that every definition and theorem we explicitly display in the following text possesses a formally proved and mechanically checked counterpart. The development is available in the Archive of Formal Proofs [14,12].

1.2 Related Work

In addition to the already mentioned work on guaranteed integration, we would like to point to related work on differential equations in theorem provers: Spitters and Makarov [16] use the constructive proof of the existence of a unique solution to calculate solutions of ODEs in Coq. The local nature of the proof restricts their computations to short existence intervals. Boldo *et al.* [2] approximate the solution of one particular partial differential equation in Coq. A formal development of Taylor models is given by Brisebarre *et al.* [4]. Platzer [23] uses differential invariants to reason about dynamical systems in a proof assistant.

1.3 Overview

Let us start with a high-level overview of our "tool": We present the required mathematical background and the formalization thereof in Section 2. Formally verified approximations of ODEs will be obtained as follows:

1. The user needs to input a term f for the right-hand side of the ODE.
2. A term for the derivative Df of f can then be obtained automatically via symbolic differentiation (Section 2.3).
3. Given f and Df, we provide a method to automatically obtain affine arithmetic approximations $\hat{f}$ and $\hat{f}'$ of f and Df. (Section 3)
4. Now f, Df $\hat{f}$, and $\hat{f}'$ can be shown to satisfy the assumptions for numerical approximations with the Euler method (Section 4).
5. Code for the Euler method can then be extracted, compiled and executed in order to obtain a list of enclosures. Theorem 9 states the correctness of the method.

We conducted experiments with some concrete ODEs in Section 5.

2 Background

We work with the interactive theorem prover Isabelle [22], inside the logic Isabelle/HOL. Isabelle is an LCF-style theorem prover, i.e., every proposition passes through a small, trusted inference kernel.

In the following, we present the background theory we use in our formalization and the notation we use in this paper to refer to it. As a subset of

Isabelle/HOL can be seen as a functional programming language, the notation we use in this presentation is inspired by functional programming languages: for a term t we write $t :: \alpha$ if t is of type α. We write function application juxtaposition as in $f\ t$ and function abstraction $\lambda x.\ t$. Types are built from base types like $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{R}^n$ or via type constructors like $\alpha \Rightarrow \beta$ for functions from type α to β, $\alpha \times \beta$ for pairs, or $\alpha\,\mathit{set}$ respectively $\alpha\,\mathit{list}$ for sets respectively lists with elements of type α. $\Rightarrow$ binds weaker than $\times$, which binds weaker than other type constructors. $\alpha\,\mathit{option}$ denotes the option type with constructors *None* and *Some*. For operations in the option monad we use Haskell-style **do**-notation. For $A :: \alpha\,\mathit{set}, B :: \beta\,\mathit{set}$ we denote with $A \to B$ the function set $\{f \mid \forall a \in A.\ f\ a \in B\}$. We make use of standard functional programming functions like *map*, *fold*, *filter*, *fst*, *snd* and write appending (concatenating) lists with $_@_ :: \alpha\,\mathit{list} \Rightarrow \alpha\,\mathit{list} \Rightarrow \alpha\,\mathit{list}$

We also make use of Isabelle's code generator [8]: it performs a (mostly syntactic) translation from equations in the logic to functions in functional programming languages like SML, OCaml, Haskell, or Scala. Worth noticing for our application is that we make use of a shallow embedding of integers $\mathbb{Z}$, i.e., operations on type $\mathbb{Z}$ are mapped to operations of the arbitrary-precision integers provided by the respective target languages.

In the remainder of this section, we present the notation of the mathematical formalization upon which we base our work.

2.1 Real Numbers

Isabelle/HOL provides a theory of real numbers $\mathbb{R}$, which does not directly allow for code generation. We formalize all of our algorithms in terms of real numbers, but in order to obtain an executable formalization, we make use of data refinement [7] and represent the type $\mathbb{R}$ with dyadic rational numbers:

We introduce (based on Obua's [21] construction of Floating point numbers) a "pseudo-constructor" $\mathit{Float} :: \mathbb{Z} \Rightarrow \mathbb{Z} \Rightarrow \mathbb{R}$ for dyadic rational numbers, i.e., $\mathit{Float}\,m\ e = m \cdot 2^e$. Isabelle's code generator can be instructed to translate $\mathbb{R}$ as a type with elements constructed by *Float* in the target language. Operations on real numbers then need to be given in terms of pattern matching on *Float*, e.g., $(\mathit{Float}\,m_1\ e_1) \cdot (\mathit{Float}\,m_2\ e_2) = \mathit{Float}\ (m_1 \cdot m2)\ (e_1 + e_2)$.

For efficiency reasons we need to restrict the precision, i.e., the size of the mantissa, during computations. We therefore use trunc^+ and trunc^- with the property $\mathit{trunc}^-\ p\ x \le x \le \mathit{trunc}^+\ p\ x$. Moreover, trunc^+ and trunc^- make sure that the absolute value of the returned mantissa is smaller than 2^p. When we speak of *precision*, we usually denote it with a value p corresponding to the length of the mantissa as described above. We also give a function *round*, for which if $\mathit{round}\,p\ x = (y, e)$, then y is rounded with precision p and $|x - y| \le e$.

Some operations like division or transcendental functions cannot be computed exactly on dyadic rational numbers, for them we use approximating functions with precision p like div^- and div^+ with $\mathit{div}^-\ p\ x\ y \le \frac{x}{y} \le \mathit{div}^+\ p\ x\ y$.

2.2 Euclidean Space

Our work is based on Isabelle/HOL's Multivariate Analysis [11], which is an extension and generalization of a port of Harrison's formalization of Euclidean space in HOL Light [9].

Euclidean spaces $\mathbb{R}^n$ are formalized as types α with a set of base vectors $\mathit{Basis} :: \alpha\ \mathit{set}$ with the vector space operations addition $+ :: \alpha \Rightarrow \alpha \Rightarrow \alpha$, scalar multiplication $\cdot :: \mathbb{R} \Rightarrow \alpha \Rightarrow \alpha$ and inner product $\bullet :: \alpha \Rightarrow \alpha \Rightarrow \mathbb{R}$. Products of real numbers are Euclidean spaces, we therefore write for example $\mathbb{R} \times \mathbb{R}$ also as $\mathbb{R}^2$, and we have e.g., $(\mathit{Basis} :: \mathbb{R}^2\ \mathit{set}) = \{(1,0),(0,1)\}$. Every element of the Euclidean space can be written as a sum of base vectors scaled with the respective coordinates. Coordinates can be extracted by taking the inner product with a base vector, so it holds that $x = \sum_{i \in \mathit{Basis}} (x \bullet i) \cdot i$.

For $a, b :: \mathbb{R}^n$ write $a \leq b$ if for all base vectors $i \in \mathit{Basis}$, $a \bullet i \leq b \bullet i$. Then the interval $[a; b] = \{x \mid a \leq x \wedge x \leq b\}$ is the smallest box containing a and b. We also define the absolute value $|a| :: \mathbb{R}^n$ componentwise, i.e., for base vectors i, $|a| \bullet i = |a \bullet i|$.

2.3 Derivatives

The (ordinary) derivative of a function $g :: \mathbb{R} \Rightarrow \mathbb{R}^n$ is written $g' :: \mathbb{R} \Rightarrow \mathbb{R}^n$. For $f :: \mathbb{R}^n \Rightarrow \mathbb{R}^m$, we denote by $\mathit{Df} :: \mathbb{R}^n \Rightarrow \mathbb{R}^n \Rightarrow \mathbb{R}^m$ the Frechet (or total) derivative of f. $\mathit{Df}\ x$ is the linear approximation of f at x (which can be represented with the Jacobian matrix). We use the notation for derivatives under the implicit assumption that they exist (which we prove or assume in the formal development).

Isabelle/HOL provides a set of rules allowing to symbolically compute derivatives. Together with the rewrite engine of Isabelle/HOL, this allows to automatically obtain a term for the derivative Df of f.

2.4 Notes on Taylor Series Expansion in Euclidean Space

In the course of formally proving the correctness of our implementation, we even identified a subtle issue in the presentation of Bouissou *et al.* [3]: They develop (in Equation 8) a Taylor series expansion of a function y, where they assume the existence of a $\xi \in [t; t+h]$ with $y(t+h) = y(t) + \sum_{i=1}^{k} \frac{h^i}{i!} y^{(i)}(t) + \frac{h^{k+1}}{k!} y^{(k+1)}(\xi)$. Such a ξ only exists for functions $y :: \mathbb{R} \Rightarrow \mathbb{R}$. In the multivariate case, $y :: \mathbb{R} \Rightarrow \mathbb{R}^n$ can be seen as a family of functions $y_i :: \mathbb{R} \Rightarrow \mathbb{R}$ such that there exists a family of $\xi_i \in [t; t+h]$ for the remainders of y_i. The remainder of y can then be written as $r := (y_i^{(k)}\ \xi_i)_{i \leq n}$. But this element need not be a member of the set $A = \{y^{(k)}(t).\ t \in [t; t+h]\}$, which they overapproximate as enclosure of the remainder in their Equation 12. However, r is an element of any box enclosing A and they use such a box enclosure in their implementation, which keeps their method safe. Consider e.g., $y(t) = (t^3 + t, t^3)$ as example illustrating the issue.

2.5 Ordinary Differential Equations

In the following, we repeat standard results about ODEs, most of which have been formalized in [13]. A homogeneous first order ODE is an equation $x'\ t = f(x\ t)$ with an unknown function $x :: \mathbb{R} \Rightarrow \mathbb{R}^n$, the independent variable t is usually denoted as time. We treat only this kind of ODE, as inhomogeneous (f may depend on t) and higher-order ODEs (only the higher derivatives of x are part of the ODE) can be reduced to the simple case. Constraining the ODE to an *initial value problem* (IVP) is crucial for the existence of a unique solution.

Definition 1 (Initial Value Problem). *An* initial value problem *ivp is a named tuple of elements $f :: \mathbb{R}^n \Rightarrow \mathbb{R}^n$, $t_0 :: \mathbb{R}$, $x_0 :: \mathbb{R}^n$, $T :: \mathbb{R}$ set, $X :: \mathbb{R}^n$ set ivp $= (f, t_0, x_0, T, X)$.*

Definition 2 (Solution). *A function $x :: \mathbb{R} \Rightarrow \mathbb{R}^n$ is a solution to an initial value problem ivp, if $x'\ t = f(x\ t)$ and $x'\ t \in X$ for all $t \in T$ and if $x\ t_0 = x_0$.*

If X is bounded, the metric space of bounded continuous functions $T \to X$ is complete. Then the Banach fixed point theorem guarantees the existence of a unique fixed point of the Picard operator $P :: (\mathbb{R} \Rightarrow \mathbb{R}^n) \Rightarrow (\mathbb{R} \Rightarrow \mathbb{R}^n)$ with $P\ x\ t = x_0 + \int_{t_0}^{t} f\ (x\ s) \mathrm{d}s$, if P is an endomorphism, i.e., maps functions from $T \to X$ onto $T \to X$.

Theorem 3 (Existence of a unique solution). *For $T = [t_0; t_1]$, if f is Lipschitz continuous (i.e., $\exists L.\ \forall x_1, x_2 \in X.\ \|f\ x_1 - f\ x_2\| \leq L \cdot \|x_1 - x_2\|$) on a compact set X and if P is an endomorphism on $T \to X$, then there exists a unique solution sol of the IVP ivp on T.*

Let us now present some results about numerical approximations of solutions. The Euler method naively approximates the solution with line segments in the direction given by the right-hand side of the ODE ($x(t + h) \in x\ t + h \cdot (f\ (x\ t)) + \mathcal{O}(h^2)$). Since the error in one step goes to zero with the stepsize h, the method is called *consistent*. We represent errors explicitly as sets, hence we give a formulation of consistency in terms of sets. We formalize enclosures of functions with the function set $X \to Y$.

Theorem 4 (Consistency of Euler method). *Assume a compact interval $[t; t + h]$, a convex and compact set $X :: \mathbb{R}^n$ set, and a function $x \in T \to X$ with derivative $x'\ t = f(x\ t)$. Further assume that f is bounded by F ($f \in X \to F$) and that the derivative Df is bounded by a box $[D_{\min}; D_{\max}]$ ($\forall x \in X.\ \forall y \in F.\ Df\ x\ y \in [D_{\min}; D_{\max}]$). Then the Euler method is consistent: $x(t + h) - x\ t + h \cdot (f(x\ t)) \in \left[\frac{h^2}{2} \cdot D_{\min}; \frac{h^2}{2} \cdot D_{\max}\right]$*

The proof makes use of the Taylor series expansion of x, which is why we assume Df bounded by a box. This ensures that the remainder (which is represented with Df) is contained in that box (cf. the discussion in Section 2.4).

3 Affine Arithmetic

We are going to adapt the Euler method to compute with sets in order to obtain a guaranteed method. We represent sets by affine forms (as described in detail in [6]) $x_0 + \sum_{i=1}^{n} \varepsilon_i \cdot x_i$, where x_0 is called the center, the x_i are coefficients and ε_i formal variables or noise symbols. The set represented by such an affine form is the set of all elements given by the form when the ε_i range in $[-1;1]$.

We represent sets α *set* with affine forms of type α *affine*. In order to stay close to an efficient executable representation, we chose α *affine* $= \mathbb{N} \times \alpha \times (\mathbb{N} \times \alpha)$ *list*. For a tuple (m, x_0, xs), $x_0 :: \mathbb{R}^n$ is the center, $xs :: (\mathbb{N} \times \mathbb{R}^n)$ *list* a list of indexed coefficients (distinct and sorted by the first component) where every index is smaller than the degree m. We write affine forms either with capital letters X, Y or explicitly as tuples. *elem* $:: \mathbb{R}^n$ *affine* $\Rightarrow (\mathbb{N} \Rightarrow \mathbb{R}) \Rightarrow \mathbb{R}^n$ returns an element given by a valuation for the formal variables: *elem* $(m, x_0, xs)\ e = \sum_{(i,x)\in xs}(e\ i) \cdot x$. *coeff* $:: (\mathbb{N} \times \mathbb{R}^n)$ *list* $\Rightarrow \mathbb{N} \Rightarrow \mathbb{R}^n$ returns the coefficient with a given index if it exists in the list and zero otherwise. The function *Affine* $:: \mathbb{R}^n$ *affine* $\Rightarrow \mathbb{R}^n$ *set* returns the set represented by an affine form: it is the set of all elements obtained via "valid" valuations: *Affine* $X = \{$*elem* $X\ e \mid e \in \mathbb{N} \to [-1;1]\}$.

An important notion is that of the joint range of affine forms. Affine forms representing the same set may have different dependencies: ε_i and ε_j represent the same set $[-1;1]$, but the subtraction $\varepsilon_i - \varepsilon_j$ represents either $\{0\}$ or $[-2;2]$, depending on whether $i = j$. More general, when reasoning about some function f taking two arguments $x \in$ *Affine* X and $y \in$ *Affine* Y, f is surely called only for arguments $(x, y) \in ($*Affine* $X) \times ($*Affine* $Y)$. But the Cartesian product discards dependencies that the affine forms are actually supposed to track: respecting the dependencies, we can be more precise and state that (x, y) is contained in the set $\{(x, y) \mid x =$ *elem* $X\ e \wedge y =$ *elem* $Y\ e \wedge e \in \mathbb{N} \to [-1;1]\}$, which is called the joint range of X and Y. We generalize this to an arbitrary number of arguments by defining the joint range for lists of affine forms via *Affines* $:: \alpha$ *affine list* $\Rightarrow \alpha$ *list set*, where we have *Affines* $xs = \{$*map* $(\lambda x.$ *elem* $x\ e)\ xs \mid e \in \mathbb{N} \to [-1;1]\}$.

The maximum deviation of an affine form (m, x_0, xs) is the sum of the absolute values of all coefficients, we denote it by *rad* $xs :: \mathbb{R}^n$. We overapproximate *rad* with precision p by safely rounding all additions: *rad*$^+$ $p\ xs =$ *fold* $(\lambda(i, x)\ e_0.$ *trunc*$^+$ $p\ (|x| + e_0))\ xs\ 0$. This can be used to obtain a bounding box for an affine form with *box* $p\ (m, x_0, xs) = [x_0 -$ *rad*$^+$ $p\ xs; x_0 +$ *rad*$^+$ $p\ xs]$, where we have *Affine* $X \subseteq$ *box* $p\ X$.

To convert boxes to affine forms, distinct noise symbols are needed for every coordinate. $[a;b]$ is represented by the affine form $\frac{a+b}{2} + \sum_{i\in \mathit{Basis}} \varepsilon_i \cdot ((\frac{b-a}{2} \bullet i) \cdot i)$, for which we write *affine-of-ivl* $a\ b$.

The Minkowski sum $A \oplus B = \{a + b.\ a \in$ *Affine* $A \wedge b \in$ *Affine* $B\}$ discards dependencies between A and B and is used for example to add some uncertainty B to a given affine form A. It can easily be implemented by adding the coefficients of B as coefficients with fresh indices to A.

We define binary coefficientwise operations that accumulate round-off errors via *round-binop* $:: \mathbb{N} \Rightarrow (\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow (\mathbb{N} \times \alpha)$ *list* $\Rightarrow (\mathbb{N} \times \alpha)$ *list* $\Rightarrow (\mathbb{N} \times \alpha)$ *list* $\times\, \alpha$. *round-binop* can be implemented efficiently thanks to the fact

that lists of coefficients are sorted. For $\textit{round-binop}\ p\ f\ xs\ ys = (zs, err)$, the first essential property is that *round-binop* distributes a binary function f rounded with precision p over the coefficients: for all $i :: \mathbb{N}$, $\textit{coeff}\ zs\ i = \textit{fst}\ (\textit{round}\ p\ (f\ (\textit{coeff}\ xs\ i)\ (\textit{coeff}\ ys\ i)))$. The second property states that err overapproximates the sum of the absolute values of all rounding errors: $\sum_{i \in \mathbb{N}} |\textit{coeff}\, zs\ i - f\ (\textit{coeff}\, xs\ i)\ (\textit{coeff}\, ys\ i)| \leq err$.

3.1 Reification of Expressions

The aim when using affine arithmetic is to replace operations in an expression on real numbers or Euclidean space by the corresponding operations on affine forms. This is similar to work by Hölzl [10] on approximations using interval arithmetic in Isabelle/HOL. This requires an explicit representation of expressions. A technique called *reification* allows to transform a term into an explicit data structure for expressions, evaluated by an interpretation function.

Let us start with expressions in real arithmetic *aexp*, for which we define an inductive datatype like in Figure 1. Elements of this datatype are interpreted recursively using the function $[\![_]\!]_{vs}$ for an environment $vs :: \mathbb{R}^n\ \textit{list}$ as given in Figure 2. The environment contains the list of free variables of the expression. They are of type $\mathbb{R}^n$ because ultimately we want to approximate functions $\mathbb{R}^n \Rightarrow \mathbb{R}^n$. $\textit{Var}\ i\ b$ allows to take the component indicated by a base vector b of the i-th element of the environment.

$$
\begin{aligned}
\textit{aexp} = {} & \textit{Add aexp aexp} \\
| \ & \textit{Mult aexp aexp} \\
| \ & \textit{Minus aexp} \\
| \ & \textit{Inverse aexp} \\
| \ & \textit{Num}\ \mathbb{R} \\
| \ & \textit{Var}\ \mathbb{N}\ \mathbb{N}
\end{aligned}
$$

Fig. 1. Inductive data type of arithmetic expressions

$$
\begin{aligned}
[\![\textit{Add}\ a\ b]\!]_{vs} &= [\![a]\!]_{vs} + [\![b]\!]_{vs} \\
[\![\textit{Mult}\ a\ b]\!]_{vs} &= [\![a]\!]_{vs} \cdot [\![b]\!]_{vs} \\
[\![\textit{Minus}\ a]\!]_{vs} &= -[\![a]\!]_{vs} \\
[\![\textit{Inverse}\ a]\!]_{vs} &= 1/[\![a]\!]_{vs} \\
[\![\textit{Num}\ r]\!]_{vs} &= r \\
[\![\textit{Var}\ i\ b]\!]_{vs} &= (vs\ !\ i) \bullet b
\end{aligned}
$$

Fig. 2. Recursive interpretation of arithmetic expressions

$$
\begin{aligned}
\textit{eexp} = {} & \textit{AddE eexp eexp} \\
| \ & \textit{Scale aexp}\ \mathbb{R}^n
\end{aligned}
$$

$$
\begin{aligned}
[\![\textit{AddE}\ x\ y]\!]_{vs} &= [\![x]\!]_{vs} + [\![y]\!]_{vs} \\
[\![\textit{Scale}\ a\ b]\!]_{vs} &= [\![a]\!]_{vs} \cdot b
\end{aligned}
$$

Fig. 3. Datatype and interpretation of Euclidean space expressions

We make use of the automated method for reification by Chaieb [5], which, given a set of equations for the interpretation function and a term, proves a reification theorem. With the equations for $[\![_]\!]$ from above and the fact that $x_2 = x \bullet b_2$ when b_2 is the second base vector, we get e.g., for the term $x_2 + 3$ the theorem $x_2 + 3 = [\![\textit{Add}\ (\textit{Var}\ 0\ b_2)\ (\textit{Num}\ 3)]\!]_{[x]}$.

For functions between Euclidean spaces, every expression $\lambda(x_1, \ldots, x_n).\ (f_1\ x_1\ \cdots\ x_n, \ldots, f_m\ x_1\ \cdots\ x_n)$ can be rewritten as $\lambda x.\ (f_1\ (x \bullet b_1)\ \cdots\ (x \bullet b_n)) \cdot b_1 + \cdots + (f_m\ (x \bullet b_1)\ \cdots\ (x \bullet b_n)) \cdot b_m$. We therefore define expression and interpretation for expressions on Euclidean space as given in Figure 3.

To give an example, the expression $(2, x_1)$ is first rewritten to $2\cdot(1,0)+(x \bullet b_1)\cdot(0,1)$ and then reified to $[\![\mathit{AddE}\ (\mathit{Scale}\ (\mathit{Num}\ 2)\ (1,0))\ (\mathit{Scale}\ (\mathit{Var}\ 0\ 1)\ (0,b_1))]\!]_{[x]}$

3.2 Approximation of Elementary Operations

For affine forms on real numbers, we support the arithmetic operations addition, multiplication, and their respective inverses. Note that, in essence, we work with a fixed, finite precision p, which means that we have to take rounding errors into account. The general approach is to round all "ideal" operations and summarize the encountered round-off errors in a fresh formal variable.

Let us illustrate this for the example of addition: We calculate the new center z with rounding error e_1, the coefficientwise addition zs of xs and ys with accumulated error e_2 and add a new coefficient (overapproximating the errors e_1 and e_2) for the formal variable with fresh index l to the resulting affine form.

$$
\begin{array}{l}
\mathit{add\text{-}affine}\ p\ (n, x_0, xs)\ (m, y_0, ys) = \\
\quad \textbf{let}\ (z, e_1) = \mathit{round}\ p\ (x_0 + y_0); \\
\quad\quad (zs, e_2) = \mathit{round\text{-}binop}\ p\ (\lambda x\ y.\ x + y)\ xs\ ys; \\
\quad\quad e = \mathit{trunc}^{+}\ p\ (e_1 + e_2); \\
\quad\quad l = \max\ n\ m \\
\quad \textbf{in}\ (l + 1, z, zs @ [(l, e)])
\end{array}
$$

Correctness of operations on affine forms states that if the arguments are members of affine sets, then the result from the "ideal" operation is in the affine set resulting from the operation on affine forms. Moreover the dependencies of the formal variables stay intact. In the example of addition:

Theorem 5 (Correctness of Addition). *If* $[x, y] \in \mathit{Affines}\ [X, Y]$, *then* $[x, y, x + y] \in \mathit{Affines}\ [X, Y, \mathit{add\text{-}affine}\ p\ X\ Y]$

We proved similar correctness theorems for multiplication *mult-affine*. multiplicative inverse *inverse-affine* and unary minus, where we guided our implementation by the descriptions in [6].

3.3 Approximation of Expressions

The explicit representation of arithmetic expressions due to reification and the approximations of elementary operations allow to recursively define an approximation function $\mathit{approx} :: \mathbb{N} \Rightarrow \mathit{aexp} \Rightarrow \mathbb{R}^n\ \mathit{affine\ list} \Rightarrow \mathbb{N} \Rightarrow \mathbb{R}\ \mathit{affine\ option}$ in affine arithmetic. Below we give addition as example but refrain from a presentation of further cases. In order not to introduce wrong dependencies, l is used as index of a fresh formal variable and needs to be threaded through the recursive calls. Approximation is performed inside the option monad, in order to handle failures like e.g., approximating the inverse of an affine form that contains zero.

$$
\begin{array}{l}
\mathit{approx}\ p\ (\mathit{Add}\ a\ b)\ vs\ l = \\
\quad \textbf{do}\ (n, x_0, xs) \leftarrow \mathit{approx}\ p\ a\ vs\ l \\
\quad\quad (m, y_0, ys) \leftarrow \mathit{approx}\ p\ b\ vs\ n \\
\quad\quad \mathit{Some}\ (\mathit{add\text{-}affine}\ p\ (n, x_0, xs)\ (m, y_0, ys))
\end{array}
$$

Approximation $\mathit{approx} :: \mathbb{N} \Rightarrow \mathit{eexp} \Rightarrow \mathbb{R}^n\ \mathit{affine\ list} \Rightarrow \mathbb{N} \Rightarrow \mathbb{R}^n\ \mathit{affine}$ of expressions in Euclidean space is just coefficientwise scaling and addition.

Correctness for the approximation of an expression in Euclidean space can then be stated as follows: If the input variables vs are in the joint range of the affine forms VS, then the approximated affine set is in the joint range with the interpreted expression. (We write $x\#xs$ for prepending the element x to the list xs)

Theorem 6 (Correctness of approximation). *If $vs \in \mathit{Affines}\ VS$, the maximum degree of the affine forms in VS is d, and $\mathit{approx}\ p\ expr\ VS\ d = \mathit{Some}\ X$, then $[\![expr]\!]_{vs}\#vs \in X\#VS$*

3.4 Summarizing Noise Symbols

During longer computations, the approximations due to affine arithmetic (and rounding errors) will add more and more noise symbols to the affine form, which impairs performance in the long run. The number of noise symbols can be reduced by summarizing (or condensing) several noise symbols into a new one. This process obviously discards the correlation mediated by the summarized noise symbols, so a trade-off needs to be found.

Following [6], we summarize all symbols with an absolute value smaller than a given fraction r (the *summarization threshold*) of the maximum deviation of the affine form. Note that we compare the coefficients in Euclidean space, that means when we summarize a noise symbol, the dependencies in all coordinates are small. We then extend the affine form consisting of the large coefficients ys with a box enclosing all small deviations zs:

$$
\begin{array}{l}
\mathit{summarize}\ p\ r\ (n, x_0, xs) = \\
\quad \textbf{let}\ rad = \mathit{rad}^+\ p\ xs \\
\quad \textbf{let}\ ys = \mathit{filter}\ (\lambda x.\ x \geq r \cdot rad)\ xs \\
\quad\quad zs = \mathit{filter}\ (\lambda x.\ \neg\ x \geq r \cdot rad)\ xs \\
\quad \textbf{in}\ (n, x_0, ys) \oplus \mathit{affine\text{-}of\text{-}ivl}\ (-\mathit{rad}^+\ p\ zs)\ (\mathit{rad}^+\ p\ zs)
\end{array}
$$

The necessary correctness theorem states that summarization returns a safe overapproximation: $\mathit{Affine}\ X \subseteq \mathit{Affine}\ (\mathit{summarize}\ p\ r\ X)$.

4 Approximation of ODEs

Our algorithm approximates ODEs in a series of discrete steps in time. We start the section by presenting the implementation and proofs for a single step, then show the extension to a series of steps.

The formalization of our algorithm and the correctness proof are generic in the ODE f, its derivative Df and respective approximations in affine arithmetic $\hat{f}$, $\hat{f}'$, which we will assume for the remainder of this section:

$f :: \mathbb{R}^n \Rightarrow \mathbb{R}^n$

$\hat{f} :: \mathbb{N} \Rightarrow \mathbb{R} \Rightarrow \mathbb{R}^n\ \mathit{affine} \Rightarrow \mathbb{R}^n\ \mathit{affine\ option}$

$\mathsf{Df} :: \mathbb{R}^n \Rightarrow \mathbb{R}^n \Rightarrow \mathbb{R}^n$
$\hat{f}' :: \mathbb{N} \Rightarrow \mathbb{R} \Rightarrow \mathbb{R}^n\ \mathsf{affine} \Rightarrow \mathbb{R}^n\ \mathsf{affine} \Rightarrow \mathbb{R}^n\ \mathsf{affine\,option}$
$\forall x.\ x \in \mathsf{Affine}\ X \longrightarrow \hat{f}\ p\ t\ X = \mathsf{Some}\ F \longrightarrow [x, f\ x] \in \mathsf{Affines}\ [X, F]$
$\forall x\ y.\ [x, y] \in \mathsf{Affines}\ [X, Y] \longrightarrow \hat{f}'\ p\ t\ X\ Y = \mathsf{Some}\ F' \longrightarrow$
$\quad [x, y, \mathsf{Df}\ x\ y] \in \mathsf{Affines}\ [X, Y, F']$
f has continuous derivative Df

4.1 Euler Step

ODEs are approximated in a series of discrete steps. A step consists of two phases, one for certification and one for approximation. In the first phase, we certify the existence of a unique solution and obtain an a-priori bound on the solution. In the second phase we use this a-priori bound to compute a tighter enclosure with a set-based Euler method. Let us assume a step size $h > 0$ at time $t_0 :: \mathbb{R}$. Further assume that the step starts at value $x_0 :: \mathbb{R}^n$, for which we assume an affine approximation X_0 with $x_0 \in \mathsf{Affine}\ X_0$.

Certification The idea is to certify the existence of a unique solution according to Theorem 3. One prerequisite is to show that the operator P used for Picard iteration is an endomorphism, which can be shown by finding a post fixed point. Like Bouissou [3], we use the set-based overapproximation $Q\ X = X_0 + h \cdot (f\ X)$ of the operator P. For a box X with $x\ t \in X$ for $t \in [t_0; t_0 + h]$, it holds that $P\ x\ t = x_0 + \int_{t_0}^{t} f\ (x\ s)\ \mathrm{d}s \in Q\ X$.

We define a function $\hat{Q}$ using affine arithmetic to overapproximate Q. Then we iterate $\hat{Q}\ p\ r$, starting with $\mathsf{box}\ p\ X_0$, until we find a post fixed point. That means when we encounter boxes B and C such that $\hat{Q}\ p\ r\ C = \mathsf{Some}\ B$ and $B \subseteq C$. Since Q is an overapproximation of P, it follows that for all $t \in T$ and $x \in T \to C$, $P\ x\ t \in C$, which certifies that P is an endomorphism on $T \to C$.

Now that we have verified P as an endomorphism, a unique solution exists according to Theorem 3, if f is Lipschitz continuous, which follows from our assumption that f is continuously differentiable.

The results of the certification phase can be summarized in the following theorem, which guarantees the existence of a unique solution for the current step size h and also provides an a-priori bound for the evolution of the solution:

Theorem 7 (Certification of Solution). *If the iteration of $\hat{Q}$ started with X_0 yields a C with $\hat{Q}\ p\ r\ C = \mathsf{Some}\ B$ and $B \subseteq C$, then the ODE f has a unique solution sol on $[t_0; t_0 + h]$ for the initial value (t_0, x_0). Moreover, the solution is bounded by $\mathsf{sol} \in [t_0; t_0 + h] \to (\mathsf{Affine}\ C)$.*

Note that it is possible that the iteration of Q does not reach a fixed point if the step size is too large – one can then repeat the phase with a smaller step size. It is also possible to accelerate the iteration with some sort of widening.

Approximation The approximation phase aims to compute a tighter enclosure for the solution, making use of the a-priori enclosure from the previous phase. For this phase we assume that the previous phase returned for some step size $h :: \mathbb{R}$ an a-priori bound $C :: \mathbb{R}^n$ *affine*.

We work with a set-based Euler method and therefore use Theorem 4, which bounds the method error of one Euler step. We first overapproximate the Euler step $\psi\ x := x + h \cdot (f\ x)$ using an affine arithmetic function $\hat{\psi}$ and add to the resulting affine form the uncertainty given by the method error.

For the overapproximation of the Euler step $\psi\ x_0$, recall the assumption $x_0 \in X_0$. We can therefore overapproximate $\psi\ x_0$ with $\hat{\psi}\ p\ r\ X_0$.

Concerning the method error, we need to bound Df. We know from the a-priori bound of Theorem 7, that for all $t \in [t_0; t_0 + h]$, $\mathit{sol}\ t \in \mathit{Affine}\ C$. We can further bound f on C with $\hat{f}$, i.e., with $F := \hat{f}\ p\ r\ C$. With the assumption that Df is bounded by $\hat{f}'$, we have for all $[x, y]$ in $\mathit{Affines}\ [C, F]$ that $[x, y, \mathit{Df}\ x\ y] \in \mathit{Affine}\ [C, F, \hat{f}'\ p\ r\ C\ F]$. If we set $[D_{\min}; D_{\max}] = \mathit{box}\ p\ (\hat{f}'\ p\ r\ C\ F)$, then Theorem 4 allows to prove that the solution is safely enclosed by an Euler step in affine arithmetic $\hat{\psi}\ p\ r\ X_0$, extended with the method error of one Euler step:

Theorem 8. $\mathit{sol}\ (t_0 + h) \in \mathit{Affine}\ (\hat{\psi}\ p\ r\ X_0 \oplus [\frac{h^2}{2} \cdot D_{\min}; \frac{h^2}{2} \cdot D_{\max}])$

4.2 Euler Series

We denote with the term "local" a solution certified by the fact that the certification phase of one Euler step succeeded. Taking the enclosure from the approximation phase (which is usually smaller, and can be made arbitrarily small with the step size) as initial enclosure X_0 for a subsequent Euler step and iterating the process, one gets a series of enclosures for local solutions. The respective step sizes are determined by the certification phase of the Euler step. Inductively, one can connect the proofs for the existence of local solutions to one theorem stating the existence of a unique global solution. The a-priori bounds can be used as bounds over local time intervals, and tight bounds can be given for discrete points in time. During computation, we accumulate several of the interval bounds and give back a list of time intervals, together with an enclosure of the solution on that interval and a tight enclosure at the end of the interval. We define the function that iterates and accumulates local steps as *euler-series* $p\ r$ for precision p and summarizing threshold r (Section 3.4), the final theorem then looks as follows:

Theorem 9. *If euler-series* $p\ r\ t_0\ X_0$ *returns* (t_1, xs)*, then there exists a unique solution on* $[t_0; t_1]$*. Moreover for all* $(t_i, C_i, t_j, X_j) \in xs$*, the solution is bounded by* C_i *resp.* X_j*: for all* $t \in [t_i; t_j]$*,* $\mathit{sol}\ t \in \mathit{Affine}\ C_i$ *and* $\mathit{sol}\ t_j \in \mathit{Affine}\ X_j$*.*

5 Experiments

Our experiments do not aim for a thorough comparison of different approaches for guaranteed integration – this can and should be done for unverified code. We

#	method	steps	time	error y	error z
1	*euler-series* 50 2^{-7}	$13 \cdot 10^3$	280 s	$1.6 \cdot 10^0$	$8.0 \cdot 10^{-2}$
2	*euler-series* 50 2^{-7}	$52 \cdot 10^3$	810 s	$2.5 \cdot 10^{-1}$	$6.0 \cdot 10^{-3}$
3	*euler-series* 50 2^{-7}	$220 \cdot 10^3$	3100 s	$6.8 \cdot 10^{-2}$	$1.6 \cdot 10^{-3}$
4	Heun [3]	$220 \cdot 10^3$	141 s	$7 \cdot 10^{-5}$	
5	ode45 [3]	$8 \cdot 10^3$	15 s	$1.7 \cdot 10^{-1}$	

Table 1. Experimental comparison for the oil reservoir problem (time interval $[0; 50]$)

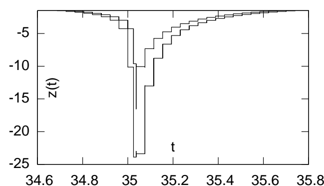

Fig. 4. Enclosures for z in the oil reservoir problem

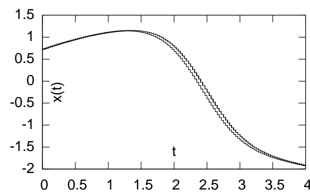

Fig. 5. Enclosures for x in $f(t,x) = x^2 - t$ with $(t_0, x_0) = (0, 0.71875)$

compare experiments using our extracted code with the experimental results of Bouissou *et al.* [3]. They run their experiments on a machine with two processors running with 2.33GHz and 2GB RAM, we perform our computations on an Intel® Core$^{\text{TM}}$2 Duo CPU T7700 at 2.40GHz and 4GB RAM.

One example they give is the oil reservoir problem $f(y,z) = (z, z^2 - \frac{3}{10^{-3}+y^2})$ for initial values $(y_0, z_0) = (10, 0)$. In Figure 4, we plot the enclosures from the list of verified bounds output by *euler-series* 50 2^{-7} when extracted to SML (4500 lines of generated code) and compiled with PolyML 5.5.1. The values are therefore verified in the sense of Theorem 9. We experience similar behavior like Bouissou *et al.* [3] in that it is hard to integrate over the time around $t = 35$, i.e., very small step sizes are used there. But also note that the method gains accuracy later on. Note that this example is not trivial, as other packages like VNODE cannot integrate this ODE.

In Table 1, we cite experimental results from Bouissou and compare with our experiments. We give the number of steps, the time needed to integrate the problem and the error of the approximation at the end of the integration. Comparing experiments with comparable step sizes, namely lines 1 and 5 resp. 3 and 4, it can be seen that our method takes roughly a factor of 20 more time. Note that the method of Heun needs twice as many evaluations of $\hat{f}$ in one step and ode45 even more. So interpret the figures just as a rough estimate, suggesting that our method is currently between one or two orders of magnitude slower than comparable tools. We believe that this is still reasonable as e.g., our method does not use native floating point numbers, where we lose a large factor. With comparable step sizes our method is less accurate, which is not surprising, as the Euler method converges linearly with the step size, the method of Heun quadratic and ode45 cubic.

A second example we would like to give is a comparison with the numerical analysis given in previous work [13]. There we integrated the ODE $f(t, x) = x^2 - t$ on the time interval $t \in [0; 0.5]$ with an error $2 \cdot 10^{-2}$. We were unable to certify the existence of a solution for a longer time span. Now (with the same computational effort of around 2 seconds), we can give enclosures for the solution on an eight times larger time span $t \in [0; 4]$ with a smaller error of $3 \cdot 10^{-3}$ at $t = 0.5$, which we consider a significant improvement.

6 Conclusion

The experiments indicate that our method exhibits reasonable performance in comparison to unverified tools and great advances when compared to previous approaches to a formally verified treatment of ODEs.

Nevertheless, there is still room for improvement: our method could be compiled for native IEEE floating point numbers, a formalization thereof is already available in Isabelle/HOL [27]. Moreover we have not yet implemented approximations of e.g., trigonometric functions, square root or the exponential function in affine arithmetic. In order to achieve competitive accuracy, methods in addition to the Euler method need to be implemented and proved consistent.

Acknowledgements

I would like to thank Olivier Bouissou for discussions on the topic. Thanks are due to Johannes Hölzl and the anonymous reviewers for valuable comments on drafts of this paper.

References

1. Berz, M., Makino, K.: Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. Reliable Computing 4(4), 361–369 (1998)
2. Boldo, S., Clément, F., Filliâtre, J.C., Mayero, M., Melquiond, G., Weis, P.: Wave equation numerical resolution: a comprehensive mechanized proof of a C program. Journal of Automated Reasoning 50(4), 423–456 (Aug 2012)
3. Bouissou, O., Chapoutot, A., Djoudi, A.: Enclosing temporal evolution of dynamical systems using numerical methods. In: Brat, G., Rungta, N., Venet, A. (eds.) NASA Formal Methods, pp. 108–123. LNCS, Springer Berlin Heidelberg (2013)
4. Brisebarre, N., Joldeş, M., Martin-Dorel, E., Mayero, M., Muller, J.m., Paşca, I., Rideau, L., Théry, L.: Rigorous Polynomial Approximation Using Taylor Models in Coq. In: Goodloe, A.E., Person, S. (eds.) NASA Formal Methods, pp. 85–99. LNCS, Springer Berlin Heidelberg (2012)
5. Chaieb, A.: Automated methods for formal proofs in simple arithmetic and algebra. Diss., Technische Universität, München (2008)
6. de Figueiredo, L.H., Stolfi, J.: Affine Arithmetic: Concepts and Applications. Numerical Algorithms 37(1-4), 147–158 (Dec 2004)

7. Haftmann, F., Krauss, A., Kunčar, O., Nipkow, T.: Data refinement in Isabelle/HOL. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving, pp. 100–115. LNCS, Springer Berlin Heidelberg (2013)
8. Haftmann, F., Nipkow, T.: Code generation via higher-order rewrite systems. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) Functional and Logic Programming (FLOPS). LNCS, vol. 6009, pp. 103–117. Springer Berlin Heidelberg (2010)
9. Harrison, J.: A HOL theory of Euclidean space. In: Hurd, J., Melham, T. (eds.) TPHOLs. LNCS, vol. 3603, pp. 114–129. Springer Berlin Heidelberg (2005)
10. Hölzl, J.: Proving inequalities over reals with computation in Isabelle/HOL. In: Reis, G.D., Théry, L. (eds.) Programming Languages for Mechanized Mathematics Systems (ACM SIGSAM '09). pp. 38–45 (2009)
11. Hölzl, J., Immler, F., Huffman, B.: Type classes and filters for mathematical analysis in Isabelle/HOL. In: Blazy, S., Christine, P.M., Pichardie, D. (eds.) Interactive Theorem Proving. LNCS, vol. 7998, pp. 279–294. Springer Berlin Heidelberg (2013)
12. Immler, F.: Affine Arithmetic. Archive of Formal Proofs (Feb 2014), `http://afp.sf.net/devel-entries/Affine_Arithmetic.shtml`
13. Immler, F., Hölzl, J.: Numerical Analysis of Ordinary Differential Equations in Isabelle / HOL. In: Beringer, L., Felty, A. (eds.) Interactive Theorem Proving. LNCS, vol. 7406, pp. 377–392. Springer Berlin Heidelberg (2012)
14. Immler, F., Hölzl, J.: Ordinary differential equations. Archive of Formal Proofs (Feb 2014), `http://afp.sf.net/devel-entries/Ordinary_Differential_Equations.shtml`
15. Lohner, R.: Einschliessung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen. Dissertation, Universität Karlsruhe (1988)
16. Makarov, E., Spitters, B.: The Picard algorithm for ordinary differential equations in Coq. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving. LNCS, vol. 7998, pp. 463–468. Springer Berlin Heidelberg (2013)
17. Nedialkov, N.S., Jackson, K.R.: The design and implementation of a validated object-oriented solver for IVPs for ODEs. Tech. rep., McMaster University (2002)
18. Nedialkov, N.S.: Interval tools for ODEs and DAEs. In: 12th GAMM-IMACS International Symposium SCAN. IEEE (2006)
19. Nedialkov, N.S.: Implementing a rigorous ODE solver through literate programming. In: Rauh, A., Auer, E. (eds.) Modeling, Design, and Simulation of Systems with Uncertainties, pp. 3–19. Mathematical Engineering, Springer (2011)
20. Neher, M., Jackson, K.R., Nedialkov, N.S.: On Taylor model based integration of ODEs. SIAM Journal on Numerical Analysis 45(1), 236–262 (Jan 2007)
21. Obua, S.: Flyspeck II: The basic linear programs. Diss., Technische Universität München, München (2008)
22. Paulson, L.C.: Isabelle: the next 700 theorem provers. Logic and Computer Science pp. 361–386 (1990)
23. Platzer, A.: The complete proof theory of hybrid systems. Logic in Computer Science (LICS) pp. 541–550 (2012)
24. Revol, N., Makino, K., Berz, M.: Taylor models and floating-point arithmetic: proof that arithmetic operations are validated in COSY. The Journal of Logic and Algebraic Programming 64(1), 135–154 (Jul 2005)
25. Stauning, O.: Automatic validation of numerical solutions. Diss., Technical University of Denmark (1997)
26. Tucker, W.: A rigorous ODE solver and Smale's 14th problem. Foundations of Computational Mathematics 2(1), 53–117 (2002)
27. Yu, L.: A Formal Model of IEEE Floating Point Arithmetic. Archive of Formal Proofs (2013), `http://afp.sf.net/entries/IEEE_Floating_Point.shtml`